Digital Image Processing Laboratory: Image Restoration

March 19, 2021

Introduction

A common inverse problem in image processing is the estimation of an image given a corrupted version. This problem is generally known as *image restoration*. One approach to this problem is to design a linear filter that predicts the desired image from the corrupted image. In Section 1, an optimal linear filter known as a *minimum mean square error* filter will be designed and applied to corrupted images.

Nonlinear filters can also be very useful in image restoration. In Section 2, a *weighted median* filter will be applied to corrupted images.

1 Minimum Mean Square Error (MMSE) Linear Filters

Often filters are designed to minimize the mean squared error between a desired image and the available noisy or distorted image. When the filter is linear, minimum mean squared error (MMSE) filters may be designed using closed form matrix expressions. Simplicity of design is an important advantage of optimal linear filters.

Suppose we are given a noisy or distorted image x and we want to estimate the image y by applying a linear filter to x. The estimate $\hat{y}_s$ at lattice location s can then be written as

$$\hat{y}_s = z_s\theta$$ - inner product

where $z_s = [x_s, x_{s+r_1}, \ldots, x_{s+r_{p-1}}]$ is a row vector of pixels from a window surrounding x_s, and θ is a column vector of filter coefficients. In MMSE filtering, the goal is to find the vector θ that will minimize the expected mean square prediction error

$$\begin{aligned} MSE &= E[|y_s - \hat{y}_s|^2] \\ \theta^* &= \arg\min_{\theta} E[|y_s - z_s\theta|^2] \ . \end{aligned}$$

The solution for θ that minimizes the MSE can be shown to be how?

$$\theta^* = R_{zz}^{-1} r_{zy}$$

Questions or comments concerning this laboratory should be directed to Prof. Charles A. Bouman, School of Electrical and Computer Engineering, Purdue University, West Lafayette IN 47907; (765) 494-0340; bouman@ecn.purdue.edu

where R_{zz} is a covariance matrix and r_{yz} is a cross correlation vector.

$$R_{zz} = E[z_s^T z_s] \quad \text{(p×p symmetric matrix)} \tag{1}$$
$$r_{zy} = E[z_s^T y_s] \quad \text{(p×1 column vector)} \tag{2}$$

In practice, the values of R_{zz} and r_{zy} may not be known, so that they must be estimated from examples of the image pairs X and Y. The coefficients for the filter may then be estimated in a training procedure known as least squares estimation. Least squares estimation determines that values of the filter coefficients which actually minimize the total squared error for a specific set of training images. To do this, let $Y = [y_1, y_2, \ldots, y_N]^T$ be a column vector of pixels from the desired image. For reasons that will be discussed later, this vector Y may *not* contain all the pixels in y. For each y_s there is an associated set of pixels $z_s = [x_s, x_{s+r_1}, \ldots, x_{s+r_{p-1}}]$ in a window surrounding x_s. We can then express the column vector of prediction errors as

$$\epsilon = Y - Z\theta$$

where

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{bmatrix} .$$

is an $N \times p$ matrix where each row z_s contains p pixels from a window surrounding the corrupted pixel x_s. The total squared error is then given by

$$\sum_{s=1}^{N} |y_s - z_s\theta|^2 = ||Y - Z\theta||^2 .$$

By differentiating, we may solve for the filter θ which minimizes the total squared error.

$$\theta^* = \hat{R}_{zz}^{-1}\hat{r}_{zy}$$ //^// b.c.

where

$$\hat{R}_{zz} = \frac{Z^T Z}{N} \tag{3}$$
$$\hat{r}_{zy} = \frac{Z^T Y}{N} . \tag{4}$$

no expectation

In practice, (3) and (4) may be too difficult to compute when all the pixels in Y are used as training samples. To reduce computational complexity, we can select a subset of locations in the image, and only train on pairs (z_s, y_s) for those selected values of s. We can express this idea formally by defining the function $\pi(i)$ to be the locations of the M selected pixels for $0 \leq i \leq M-1$. The vector $Y = [y_{\pi(1)}, y_{\pi(2)}, \ldots, y_{\pi(M)}]^T$ is then a column vector of pixels in y at the selected locations. The corresponding matrix

$$Z = \begin{bmatrix} z_{\pi(1)} \\ z_{\pi(2)} \\ \vdots \\ z_{\pi(M)} \end{bmatrix}$$

is then formed by the associated windows in x centered about the locations $x_{\pi(i)}$. Notice that the original images x and y are left at their original resolution, but that the pair of (z_s, y_s) are sparsely sampled.

1. Down load the zip file *images.zip* from the lab home page. This file contains an image, *img14g.tif*, a blurred version *img14bl.tif*, and two noisy versions, *img14gn.tif* and *img14sp.tif*.

2. Use Python to compute estimates of the covariance matrix $\hat{R}_{zz}$ and the cross correlation $\hat{r}_{zy}$ for a 7×7 prediction window. Use the original *img14g.tif* for Y and use *img14bl.tif* for X. Only sample the pairs (z_s, y_s) at (1/400)th of the pixel locations in the image. You can do this by taking a sample at every 20th column and every 20th row. The Python *numpy.reshape* command may be useful in this exercise.

3. Using your estimates $\hat{R}_{zz}$ and $\hat{r}_{zy}$, compute the corresponding filter coefficients θ^*.

4. Apply the optimal filter to the image *img14bl.tif*. Print or export the filtered image for your report.

5. Repeat this procedure using *img14gn.tif* for X. Then repeat the procedure using *img14sp.tif* for X.

Section 1 Report:

1. Hand in the four original images *img14g.tif, img14bl.tif, img14gn.tif* and *img14sp.tif*.

2. Hand in the output of the optimal filtering for the blurred image and the two noisy images.

3. Hand the MMSE filters that you computed for the blurred image and the two noisy images. (Each filter is specified by the optimum value of θ^* that you calculated.) Orient each filter into a 7x7 array to make the spatial orientation clear. For each filter, clearly state which corrupted image was used to compute the filter coefficients, θ^*.

2 Weighted Median Filtering

A simple median filter is a nonlinear filter which simply replaces each pixel with the median from a set in a window surrounding the pixel. This has the effect of minimizing the absolute prediction error. The output of the filter can therefore be written as the following

$$Y_s = \arg\min_{\theta} \sum_{k \in w(s)} |X_k - \theta| \tag{5}$$

where $w(s)$ is a window surrounding pixel s.

A weighted median filter allows some pixels in the window to have more influence on the output than others. Here, the output is written as

$$Y_s = \arg\min_{\theta} \sum_{k \in w(s)} a_{s-k} |X_k - \theta| \tag{6}$$

where a_{s-k} are weighting factors which determine the relative influence pixels in $w(s)$ have on the output. A typical set of weights is shown in Figure 1. This example allows the pixels closer to the current pixel to have a stronger influence on the output.

1	1	1	1	1
1	2	2	2	1
1	2	2	2	1
1	2	2	2	1
1	1	1	1	1

Figure 1: Weighting factors

The weighted median is found by sorting the pixels in the window in descending order. The corresponding pixel weights are placed in the same order as the sorted pixels.

$$\left\{X_{(1)}, X_{(2)}, \ldots, X_{(p)}\right\}$$

$$\left\{a_{(1)}, a_{(2)}, \ldots, a_{(p)}\right\}$$

Then the weighted median $X_{(i^*)}$ is determined by incrementing the index i^* until the following holds true.

$$\sum_{i=1}^{i^*} a(i) \geq \sum_{i^*+1}^{p} a(i) \tag{7}$$

1. Write a C program that will apply a 5×5 weighted median filter to an image, using the weights in figure 1.

2. Down load the zip file *images.zip* from the lab home page. This file contains an image, *img14g.tif*, and two corrupted versions, *img14gn.tif* and *img14sp.tif*.

3. Apply the median filter to these two noisy images, and compare the results to the original image. Print or export the results for your report.

Section 2 Report:

1. Hand in your results of median filtering.

2. Hand in your C code.